2301 Assignment 1 Group 10

[Chu Gong, Zhihao Ren, Anthony Voyage, Kaixi Wu] Group member.

1.

(1) $MPR = \frac{5.25\%}{12}$

$$\frac{P}{MPR} + \frac{P}{MPR^2} + \cdots + \frac{P}{(MPR)^{360}} = 1000000$$

$\Rightarrow P = 5522.04$

(b). $$\frac{P}{(MPR)^1} + \frac{P}{(MPR)^2} + \cdots + \frac{P}{(MPR)^{240}} = 856495.04.$$

(c): calculate the loan value at payment 108. and payment 120.

$$loan\_reduce = -\left(\frac{P}{MPR^1} + \cdots + \frac{P}{MPR^{240}}\right) + \left(\frac{P}{MPR^1} + \cdots + \frac{P}{MPR^{252}}\right)$$

$\therefore$ interest $= \underbrace{P \times 12}_{\text{net pay}} -$ loan reduce

$\Rightarrow$ interest $= 43670.50$

2.

(a): $f(0,0,1) = r_1(0,1) = 5\%$

For $f_1(0,1,2)$:

$$\frac{1}{(1+r_1(0,2))^2} = \frac{1}{(1+r_1(0,1))} \cdot \frac{1}{(1+f_1(0,1,2))}$$

$\Rightarrow f_1(0,1,2) = 7.01\%$

For $f_1(0,2,3)$

$$\frac{1}{(1+r_1(0,3))^3} = \frac{1}{(1+r_1(0,1))^2} \cdot \frac{1}{(1+f_1(0,2,3))}$$

$\Rightarrow f_1(0,2,3) = 9.03\%$

(b): Use $r_1(0,1)$ and $r_1(0,3)$.

You lend $\frac{1000}{(1+r_1(0,1))} = \$952$ today.

Then you will get $\$952 \times (1+r_1(0,3))^3 = 1166.7$ at T=3 year.
which is the same as the money that
you need to pay back at $T=3$.
which is $1000 \times (1+f_1(0,1,3))^2 = 1166.7$

Q3

$r_1(0,1) = f_1(0,0,1)$

$r_1(0,2) = [(1 + r_1(0,1))(1 + f_1(0,1,2))]^{1/2} - 1$

$r_1(0,3) = [(1 + r_1(0,2))^2(1 + f_1(0,2,3))]^{1/3} - 1$

$\Rightarrow$

$r_1(0,1) = 0.11, r_1(0,2) = 0.12, r_1(0,3) = 0.1364$

Q4

For treasury bill, the price is quoted as discount, and we have: $n = 32, d = 5.2\%$, therefore:
$P = 10000\left(1 - \dfrac{nd}{360}\right) = 9953.7778$

5 Question 5

A discount bond has price $<$ face value. YTM is the effective single discount rate across all cash flows associated with the bond that would result in the market price. Current yield is an investment's annual income divided by its current market price. If we suppose (and with no loss of generality) that bonds have a face value of $\$100$, market price P and coupon rate $c = \frac{C}{100}$ then current yield = $\frac{c\times 100}{P}$.
We know that a bond trading at par will have yield to maturity equal to the coupon rate, $y = c$. Suppose that there is such a bond. Let's call this yield to maturity y_1. Now suppose that we have another bond that pays coupons at a rate $c = \frac{C}{P}$ and also trades at par. Again, we must have that the yield of this bond, y_2, is equal to the coupon rate $\frac{C}{P}$. If we linearly scale all cashflows in this second bond we will not affect its yield. We could use as a linear factor $\frac{100}{P}$. This new bond will still have yield = $y_2 = \frac{C}{P}$, however now it must be the case that the final repayment is not $\$100$ but instead P and coupon rate is again c. The price of this bond will now be P though. So now we can make some direct between our original bond and a bond trading at a discount, price P, and paying coupons C with face value of 100. The yield on this bond must be $> \frac{C}{P}$. as the final repayment of this bond is greater than P, and we just showed that the yield on a bond with the same features but final repayment P is $\frac{C}{P}$. Hence the yield to maturity must be greater than the current yield for a discount bond.
Finally, so long as $C > 0$ and the bond is trading at discount then the current yield must always be greater than the coupon rate and the current yield is comprised of the sum of all coupon payments plus the final repayment, all discounted back at their respective rates. Again using a face value of $\$100$ the coupon rate is $\frac{c\times 100}{100}$ and as $P < 100$ for a discount bond then clearly the coupon rate is lower than the current yield.
So, in two parts we have shown that YTM $>$ current yield and current yield $>$ coupon rate.

6 Question 6

Rates for this question are displayed in Figure 1. Note that the yield curve for 29 May 2018 will have settlement date 31 May 2018.

Date	Zero Rate	Forward Rate
05/31/2018	—	2.31813000000000
08/31/2018	2.37666301585850	2.33346095337569
11/30/2018	2.37150478655406	2.46702480555569
02/28/2019	2.42393753171055	2.55633514771236

Figure 1: Bloomberg "23" yield curve for trade date May 29, 2018

6.1 6a

Discount factors from zero rates are calculated using the 30I/360 day-count convention, compounded semi-annually.

$$1 + \text{interest} = \left(1 + \frac{\text{quoted rate}}{2}\right)^{2\times \frac{\text{(30I/360) days in interest period}}{360}}$$

Here:

$$\begin{aligned}
(D_1.M_1.Y_1) &= (31, 5, 2018) \rightarrow D_1 = 30 \\
(D_2.M_2.Y_2) &= (28, 2, 2019) \rightarrow D_2 = 28 \\
\text{Day count under convention} &= 360(2019 - 2018) + 30(2 - 5) + (28 - 30) \\
&= 268
\end{aligned}$$

And so now the discount factor can be calculated as

$$\begin{aligned}
DF_{2/28/2019} &= \frac{1}{\left(1 + \frac{0.0242393753171055}{2}\right)^{\left(2\times\frac{268}{360}\right)}} \\
&= 0.98222350492166
\end{aligned}$$

6.2 6b

Discount factors from forwards rates are calculated using the Actual/360 day-count convention, compounded quarterly.

$$\text{Interest} = \text{quoted rate} \times \frac{\text{actual days in interest period}}{360}$$

We can find the overall discount factor from finding the intervening quarterly discount factors[1] and multiplying together.

$$\begin{aligned}
DF_{2/28/2019} &= DF_{31/8/2018} \times DF_{30/11/2018} \times DF_{2/28/2019} \\
&= \frac{1}{1 + (0.0231813000000000 \times \frac{92}{360})} \times \frac{1}{1 + (0.0233346095337569 \times \frac{91}{360})} \times \frac{1}{1 + (0.0246702480555569 \times \frac{90}{360})} \\
&= 0.99411077839659 \times 0.99413611719790 \times 0.99387024363860 \\
&= 0.98222350492166
\end{aligned}$$

[1] n.b. we are slightly loose with the subscripting notation in the equations here. $DF_{2/28/2019}$ is the discount factor for 28 February 2019 based on rates at 31 May 2018. The other discount factors are quarterly discount factors. E.g $DF_{30/11/2018}$ is the discount factor based on quarterly forward rates available at 29 May 2018 for contracts beginning 31 August 2018.

Q7 Fitting yield curve

In [1]:
```
import numpy as np
import pandas as pd
import copy
from scipy import optimize
from scipy.optimize import minimize
from pandas.tseries.holiday import USFederalHolidayCalendar
from pandas.tseries.offsets import CustomBusinessDay

us_bd = CustomBusinessDay(calendar=USFederalHolidayCalendar())


from pandas.tseries.offsets import DateOffset

class BaseYieldCurve:
    def __init__(self, input_date, start_date, input_path, end_date):
        self.input_date = input_date
        self.start_date = start_date
        self.end_date = end_date
        self.input_data = pd.read_excel(input_path)

        self.fwd_dates = pd.Series(dtype=object)
        self.swap_dates = pd.Series(dtype=object)

        self._input_preprocess()
        self._initialize_output_df() # define output dates list


    def _input_preprocess(self):
        # calculate some required columns
        self.input_data["Mid"] = (self.input_data["Bid"]+self.input_data["Ask"])/2
        self.input_data = self.input_data[["Term", "Unit", "Rate Type", "Daycount", "Freq", "Mid"]]

        self.input_data["Maturity_date"] = self.input_data.apply(
            lambda r: self._get_maturity_date(r), axis=1)

        # append interpolate maturity dates
        self._append_FRA_dates()
        self._append_swap_dates()

        self.input_data["ACT_days"] = (pd.to_datetime(self.input_data["Maturity_date"]) -
                                       pd.to_datetime(self.start_date)).dt.days
```

```
        self.input_data["30I_days"] = self.input_data.apply(lambda r: self._get_30I_days(r), axis=1)

        self.input_data["ACT/360"] = self.input_data["ACT_days"]/360.
        self.input_data["30I/360"] = self.input_data["30I_days"]/360.

        self.input_data["discount_factor"] = np.nan
        self.input_data["simple_rate"] = np.nan

        # initialize at start date (first row)
        if self.input_data.loc[0, "Maturity_date"] == self.start_date:
            self.input_data.loc[0, "Rate Type"] = "start"
            self.input_data.loc[0, "discount_factor"] = 1.0
            self.input_data.loc[0, "simple_rate"] = 0.0
            self.input_data.loc[0, "Mid"] = 0.0

    def _append_FRA_dates(self):
        # FRA effective dates (Third Wed of the month)
        df_FRA_dates = pd.DataFrame(
            pd.date_range(
                start=self.start_date,
                end=self.input_data.loc[
                    self.input_data["Rate Type"]=="Contiguous Futures", "Maturity_date"].values[-1]
            ), columns=["date"]
        )
        df_FRA_dates["weekday"] = df_FRA_dates["date"].dt.weekday+1
        df_FRA_dates["year"] = df_FRA_dates["date"].dt.year
        df_FRA_dates["month"] = df_FRA_dates["date"].dt.month
        df_FRA_dates = df_FRA_dates.loc[(df_FRA_dates["month"]%3 == 0) & (df_FRA_dates["weekday"] == 3)]

        df_FRA_dates["rank"] = df_FRA_dates.groupby(by=["year", "month"])["date"].rank()
        df_FRA_dates = df_FRA_dates.loc[df_FRA_dates["rank"]==3]
        df_FRA_dates["date"] = df_FRA_dates["date"].dt.strftime("%Y%m%d")
        self.fwd_dates = df_FRA_dates["date"].reset_index(drop=True)

        for mat_date in set(df_FRA_dates["date"]) - set(self.input_data["Maturity_date"]):
            self.input_data = self.input_data.append(
                {"Maturity_date":mat_date, "Rate Type":"interpolate"}, ignore_index=True)
        self.input_data = self.input_data.sort_values(by="Maturity_date").reset_index(drop=True)

    def _append_swap_dates(self):
        # semiannually coupon payment (and also business day adjust)
```

```
        df_swap_dates = pd.DataFrame(
            pd.date_range(
                start=self.start_date,
                end=self.input_data.loc[self.input_data["Rate Type"]=="Swap Rates", "Maturity_date"].values[-
1]
            ), columns=["date"]
        )

        df_swap_dates["year"] = df_swap_dates["date"].dt.year
        df_swap_dates["month"] = df_swap_dates["date"].dt.month
        df_swap_dates["day"] = df_swap_dates["date"].dt.day
        df_swap_dates = df_swap_dates.loc[
            (df_swap_dates["day"]==int(self.start_date[6:]))
            & (df_swap_dates["month"].isin(
                [int(self.start_date[4:6]), int(self.start_date[4:6])+6]))]
        # df_swap_dates["date"] = df_swap_dates["date"].apply(
        #     lambda x: pd.bdate_range(start=x, periods=1)[0].strftime("%Y%m%d"))
        df_swap_dates["date"] = df_swap_dates["date"].apply(
            lambda x: pd.date_range(start=x, periods=1, freq=us_bd)[0].strftime("%Y%m%d"))
        self.swap_dates = df_swap_dates["date"].reset_index(drop=True) # save this for following curve fittin
g

        for mat_date in set(df_swap_dates["date"]) - set(self.input_data["Maturity_date"]):
            self.input_data = self.input_data.append(
                {"Maturity_date":mat_date, "Rate Type":"interpolate"}, ignore_index=True)
        self.input_data = self.input_data.sort_values(by="Maturity_date").reset_index(drop=True)

    def _get_maturity_date(self, temp_r):
        # use Term and Unit to calculate maturity dates list
        # need to adjust for business day

        if temp_r["Unit"] == "MO":
            shift_date = pd.to_datetime(self.start_date) + DateOffset(months=int(temp_r["Term"]))
            # maturity_date = pd.bdate_range(start=shift_date, periods=1)[0].strftime("%Y%m%d")
            maturity_date = pd.date_range(start=shift_date, periods=1, freq=us_bd)[0].strftime("%Y%m%d")
        elif temp_r["Unit"] == "ACTDATE":
            maturity_date = str(temp_r["Term"])[:8]  # int to str yyyymmmdd
        elif temp_r["Unit"] == "YR":
            shift_date = pd.to_datetime(self.start_date) + DateOffset(years=int(temp_r["Term"]))
            # maturity_date = pd.bdate_range(start=shift_date, periods=1)[0].strftime("%Y%m%d")
            maturity_date = pd.date_range(start=shift_date, periods=1, freq=us_bd)[0].strftime("%Y%m%d")
        else:
```

```
            raise Exception("Unit type do not implemented.")
        return maturity_date

    def _get_30I_days(self, temp_r):
        # 30I days, from start date to maturity date
        start_year, start_month, start_day = int(self.start_date[:4]), \
                    int(self.start_date[4:6]), int(self.start_date[6:])
        # check is last day of the month?
        if start_day == 31:
            start_day = 30

        mat_year, mat_month, mat_day = int(temp_r["Maturity_date"][:4]), \
                    int(temp_r["Maturity_date"][4:6]), int(temp_r["Maturity_date"][6:])
        if mat_day == 31:
            mat_day = 30

        return 360*(mat_year-start_year) + 30*(mat_month-start_month) + (mat_day-start_day)

    def _initialize_output_df(self):
        df_output = pd.DataFrame(
            pd.date_range(start=self.start_date,
                          end=max(self.end_date, self.input_data["Maturity_date"].iloc[-1])),
            columns=["Maturity_date"]
        )
        df_output["month"] = df_output["Maturity_date"].dt.month
        df_output["day"] = df_output["Maturity_date"].dt.day
        df_output = df_output.loc[(df_output["day"]==int(self.start_date[6:]))
            & ((df_output["month"]-int(self.start_date[4:6]))%3==0)]
        # df_output["Maturity_date"] = df_output["Maturity_date"].apply(
        #    lambda x: pd.bdate_range(start=x, periods=1)[0].strftime("%Y%m%d"))
        df_output["Maturity_date"] = df_output["Maturity_date"].apply(
            lambda x: pd.date_range(start=x, periods=1, freq=us_bd)[0].strftime("%Y%m%d"))
        df_output = df_output[["Maturity_date"]].reset_index(drop=True)

        df_output["ACT_days"] = (pd.to_datetime(df_output["Maturity_date"]) -
                                 pd.to_datetime(self.start_date)).dt.days
        df_output["30I_days"] = df_output.apply(lambda r: self._get_30I_days(r), axis=1)
        df_output["ACT/360"] = df_output["ACT_days"]/360.
        df_output["30I/360"] = df_output["30I_days"]/360.

        df_output["is_keep"] = 1
```

```
        self.df_output = df_output

    def fit_curve(self):
        pass
```

In [2]:
```
class PiecewiseLinearYieldCurve(BaseYieldCurve):
    def __init__(self, input_date, start_date, input_path, end_date):
        BaseYieldCurve.__init__(self, input_date, start_date, input_path, end_date)

    def fit_curve(self):

        # start from LIBOR
        cond_libor_discount = (self.input_data["Rate Type"]=="Cash Rates")
        self.input_data.loc[cond_libor_discount, "discount_factor"] = \
            1/(1+(self.input_data.loc[cond_libor_discount, "Mid"]/100)*
               self.input_data.loc[cond_libor_discount, "ACT/360"])
        self.input_data.loc[cond_libor_discount, "simple_rate"] = \
            ((1/self.input_data.loc[cond_libor_discount, "discount_factor"])-1)/ \
                (self.input_data.loc[cond_libor_discount, "ACT/360"])

        # min range (any required maturity that less than 3m LIBOR)
        min_start_idx = 1  # skip start date
        min_end_idx = self.input_data.loc[self.input_data["Mid"].notnull()].index[1]  # first non-nan asset
        self.input_data.loc[min_start_idx:(min_end_idx-1), "simple_rate"] = \
            self.input_data.loc[min_end_idx, "simple_rate"]

        self.input_data.loc[min_start_idx:(min_end_idx-1), "discount_factor"] = \
            1/(1+(self.input_data.loc[min_start_idx:(min_end_idx-1), "simple_rate"])*
               self.input_data.loc[min_start_idx:(min_end_idx-1), "ACT/360"])

        # linear interpolation for forwards
        cond_fwd = (self.input_data["Rate Type"]=="Contiguous Futures")
        for fwd_end_date in self.input_data.loc[cond_fwd, "Maturity_date"]:

            fwd_end_idx = self.input_data.loc[
                self.input_data["Maturity_date"]==fwd_end_date].index[0]

            fwd_start_date = self.fwd_dates[self.fwd_dates<fwd_end_date].iloc[-1]
            fwd_start_idx = self.input_data.loc[
                self.input_data["Maturity_date"]==fwd_start_date].index[0]

            fwd_act_time = self.input_data.loc[fwd_end_idx, "ACT/360"] - \
                                        self.input_data.loc[fwd_start_idx, "ACT/360"]
            spot_discount = self.input_data.loc[fwd_start_idx, "discount_factor"]
            fwd_discount = 1/(1+(self.input_data.loc[fwd_end_idx, "Mid"]/100)*fwd_act_time)
            self.input_data.loc[fwd_end_idx, "discount_factor"] = spot_discount * fwd_discount
```

```
        self.input_data.loc[cond_fwd, "simple_rate"] = \
            ((1/self.input_data.loc[cond_fwd, "discount_factor"])-1)/self.input_data.loc[cond_fwd, "ACT/360"]

        # nans in forwards maturity range
        cond_fwd_nans =
        (self.input_data["Maturity_date"]>=self.input_data.loc[cond_fwd, "Maturity_date"].values[0]) \
        & (self.input_data["Maturity_date"]<=self.input_data.loc[cond_fwd, "Maturity_date"].values[-1]) \
        & (self.input_data["Rate Type"]=="interpolate")

        for fwd_nan_idx in self.input_data.loc[cond_fwd_nans].index:
            prev_idx = self.input_data.loc[
                (self.input_data.index<fwd_nan_idx) & (self.input_data["simple_rate"].notnull())
            ].index[-1]

            next_idx = self.input_data.loc[
                (self.input_data.index>fwd_nan_idx) & (self.input_data["simple_rate"].notnull())
            ].index[0]

            total_rate = (self.input_data.loc[next_idx, "simple_rate"] -
                                self.input_data.loc[prev_idx, "simple_rate"])
            total_period = (self.input_data.loc[next_idx, "ACT/360"] -
                                self.input_data.loc[prev_idx, "ACT/360"])
            delta_period = (self.input_data.loc[fwd_nan_idx, "ACT/360"] -
                                self.input_data.loc[prev_idx, "ACT/360"])

            self.input_data.loc[fwd_nan_idx, "simple_rate"] = total_rate * delta_period/total_period + \
                            self.input_data.loc[prev_idx, "simple_rate"]

        self.input_data.loc[cond_fwd_nans, "discount_factor"] = \
            1/(1+(self.input_data.loc[cond_fwd_nans, "simple_rate"])*
               self.input_data.loc[cond_fwd_nans, "ACT/360"])

        # linear interpolation for swaps
        # start with first swap
        swap_end_date = self.input_data.loc[self.input_data["Rate Type"]=="Swap Rates", "Maturity_date"].iloc
[0]
        swap_end_idx = self.input_data.loc[self.input_data["Maturity_date"]==swap_end_date].index[0]
        coupon_dates = self.swap_dates[self.swap_dates<=swap_end_date][1:]
        swap_yield = (self.input_data.loc[swap_end_idx, "Mid"]/100)
        freq = self.input_data.loc[swap_end_idx, "Freq"]
```

```
        df_coupons = copy.deepcopy(self.input_data.loc[
            self.input_data["Maturity_date"].isin(coupon_dates)])
        swap_discount = (1-(swap_yield/freq)*(df_coupons["discount_factor"][:-1].sum()))/(1+(swap_yield/freq
))
        self.input_data.loc[swap_end_idx, "discount_factor"] = swap_discount
        self.input_data.loc[swap_end_idx, "simple_rate"] = \
                ((1/swap_discount)-1)/self.input_data.loc[swap_end_idx, "ACT/360"]

        # solve following by optimize (find root)
        swap_data_idxs = self.input_data.loc[self.input_data["Rate Type"]=="Swap Rates"].index
        # for swap_end_date in self.input_data.loc[self.input_data["Rate Type"]=="Swap Rates", "Maturity_dat
e"].iloc[1:]:
        for i, swap_end_idx in enumerate(swap_data_idxs[1:]):
            # swap_end_idx = self.input_data.loc[self.input_data["Maturity_date"]==swap_end_date].index[0]
            swap_prev_idx = swap_data_idxs[i]
            # print(swap_prev_idx, swap_end_idx)

            swap_end_date = self.input_data.loc[swap_end_idx, "Maturity_date"]

            coupon_dates = self.swap_dates[self.swap_dates<=swap_end_date][1:]
            swap_yield = (self.input_data.loc[swap_end_idx, "Mid"]/100)
            freq = self.input_data.loc[swap_end_idx, "Freq"]

            df_coupons = copy.deepcopy(self.input_data.loc[
                self.input_data["Maturity_date"].isin(coupon_dates)])
            df_coupons["coupon_interval"] = df_coupons["30I/360"] - df_coupons["30I/360"].shift(1).fillna(0.)
            total_period = df_coupons.loc[swap_end_idx, "ACT/360"]-df_coupons.loc[swap_prev_idx, "ACT/360"]
            start_rate = df_coupons.loc[swap_prev_idx, "simple_rate"]
            # delta_period = df_coupons.loc[swap_end_idx-1, "ACT/360"]-df_coupons.loc[swap_end_idx-2, "ACT/36
0"]


            def swap_func(rate_x):
                final_rate = rate_x
                total_rate = rate_x-start_rate

                # coupon_discount_sum = df_coupons.loc[:swap_prev_idx, "discount_factor"].sum()
                coupon_discount_sum = (df_coupons.loc[:swap_prev_idx, "discount_factor"] *
                                       df_coupons.loc[:swap_prev_idx, "coupon_interval"]).sum()
                for idx_coupon in range(swap_prev_idx+1, swap_end_idx):
                    delta_period = df_coupons.loc[idx_coupon, "ACT/360"]-df_coupons.loc[swap_prev_idx, "ACT/3
60"]
                    temp_rate = total_rate * delta_period / total_period + start_rate
```

```
                    temp_discount = 1/(1+(temp_rate)*df_coupons.loc[idx_coupon, "ACT/360"])
                    coupon_discount_sum += temp_discount * df_coupons.loc[idx_coupon, "coupon_interval"]

                final_discount = 1/(1+(final_rate)*df_coupons.loc[swap_end_idx, "ACT/360"])
                # coupon_discount_sum += final_discount
                coupon_discount_sum += final_discount * df_coupons.loc[swap_end_idx, "coupon_interval"]
                return final_discount + coupon_discount_sum * swap_yield -1
                # return final_discount + coupon_discount_sum * swap_yield/freq -1

            res = optimize.root(swap_func, [0.0])
            self.input_data.loc[swap_end_idx, "simple_rate"] = res.x[0]
            total_rate = self.input_data.loc[swap_end_idx, "simple_rate"] - start_rate
            for idx_coupon in range(swap_prev_idx+1, swap_end_idx):
                delta_period = df_coupons.loc[idx_coupon, "ACT/360"]-df_coupons.loc[swap_prev_idx, "ACT/360"]
                self.input_data.loc[idx_coupon, "simple_rate"] = total_rate * delta_period / total_period + s
tart_rate
            # self.input_data.loc[swap_end_idx-1, "simple_rate"] = start_rate + \
            #     (self.input_data.loc[swap_end_idx, "simple_rate"]-start_rate)*(delta_period / total_period)

            # also fill the discount factor for next iteration
            self.input_data.loc[(swap_prev_idx+1):swap_end_idx, "discount_factor"] = \
                1/(1+(self.input_data.loc[(swap_prev_idx+1):swap_end_idx, "simple_rate"])*
                   self.input_data.loc[(swap_prev_idx+1):swap_end_idx, "ACT/360"])

            # print(swap_end_date, res.x[0])

        # max range

    def get_final_output(self):
        df_final_output = pd.concat([self.df_output,
            self.input_data[
                ["Maturity_date", "discount_factor", "simple_rate", 'ACT_days', '30I_days', 'ACT/360', '30I/3
60']]
                  ], axis=0)
        df_final_output = df_final_output.sort_values(by=["Maturity_date", "discount_factor"]).reset_index(dr
op=True)
        df_final_output["is_keep"] = df_final_output.groupby(by="Maturity_date")["is_keep"].bfill().values
        df_final_output = df_final_output.drop_duplicates(subset=["Maturity_date"], keep="first").reset_index
(drop=True)

        for idx in df_final_output.loc[df_final_output["simple_rate"].isnull()].index:
            total_rate = df_final_output.loc[idx+1, "simple_rate"] - df_final_output.loc[idx-1, "simple_rate"
```

```
]
            total_period = df_final_output.loc[idx+1, "ACT/360"] - df_final_output.loc[idx-1, "ACT/360"]

            delta_period = df_final_output.loc[idx, "ACT/360"] - df_final_output.loc[idx-1, "ACT/360"]
            df_final_output.loc[idx, "simple_rate"] = df_final_output.loc[idx-1, "simple_rate"] + \
                    total_rate * delta_period / total_period

        cond_nan_discount = df_final_output["discount_factor"].isnull()
        df_final_output.loc[cond_nan_discount, "discount_factor"] = \
                1/(1+(df_final_output.loc[cond_nan_discount, "simple_rate"])*
                      df_final_output.loc[cond_nan_discount, "ACT/360"])

        df_final_output = df_final_output.loc[df_final_output["is_keep"]==1].reset_index(drop=True)

        fwd_discount = df_final_output["discount_factor"].shift(-1) / df_final_output["discount_factor"]
        df_final_output["forward_rate"] = ((1/fwd_discount)-1)/ \
                (df_final_output["ACT/360"].shift(-1) - df_final_output["ACT/360"])
        df_final_output["forward_rate"] = df_final_output["forward_rate"]*100

        # semi-annually compounded zero rate
        df_final_output["zero_rate"] = (np.power((1/df_final_output["discount_factor"]),
                                        1/(df_final_output["30I/360"]*2))-1)*2
        df_final_output["zero_rate"] = df_final_output["zero_rate"]*100

        return df_final_output
```

In [ ]:

In [3]:
```
# input_path = "bbg_curve_input_022819.xlsx"
# base_yield_curve = BaseYieldCurve("20190228", "20190304", input_path)

input_path = "libor_rates_input_022819.xlsx"
input_date = "20190228"
start_date = "20190304"
end_date = "20681204"

simple_yield_curve = PiecewiseLinearYieldCurve(input_date, start_date, input_path, end_date)
simple_yield_curve.fit_curve()

df_final_output = simple_yield_curve.get_final_output()
```

In [4]:
```
# compare BBG output
# df_BBG_output = pd.read_excel("bbg_curve_output_022819.xlsx")
df_BBG_output = pd.read_excel("libor_rates_output_022819.xlsx")
df_BBG_output = df_BBG_output
df_BBG_output.columns = ["{}(BBG)".format(c) for c in df_BBG_output.columns]

df_final_output_table = pd.concat([df_final_output, df_BBG_output], axis=1)
df_final_output_table = df_final_output_table[
    ["Maturity_date", "discount_factor", "zero_rate", "Zero Rate(BBG)",
    "forward_rate", "Forward Rate(BBG)",]]

df_final_output_table["Zero Rate Error"] = df_final_output_table["zero_rate"] - df_final_output_table["Zero R
ate(BBG)"]
df_final_output_table["Forward Rate Error"] = df_final_output_table["forward_rate"] - df_final_output_table[
"Forward Rate(BBG)"]
```

In [6]:
```
df_final_output_table
```

Out[6]:

	Maturity_date	discount_factor	zero_rate	Zero Rate(BBG)	forward_rate	Forward Rate(BBG)	Zero Rate Error	Forward Rate Error
0	20190304	1.000000	0.000000	0.000000	2.615130	2.615130	0.000000e+00	-8.304468e-14
1	20190604	0.993361	2.682177	2.682177	2.592484	2.592484	-9.059420e-14	-4.440892e-15
2	20190904	0.986823	2.670525	2.670525	2.607773	2.607773	-4.352074e-14	3.996803e-15
3	20191204	0.980361	2.662162	2.662162	2.632146	2.632146	-4.662937e-14	-8.926193e-14
4	20200304	0.973881	2.664183	2.664183	2.578889	2.578889	-4.440892e-14	-8.437695e-14
...	...	...	...	...	...	...	...	...
196	20680305	0.256425	2.796603	2.796603	2.374846	2.374895	2.220446e-15	-4.887737e-05
197	20680604	0.254895	2.794789	2.794789	2.369566	2.369566	-2.664535e-15	-5.817569e-14
198	20680904	0.253360	2.792943	2.792943	2.364141	2.364141	8.881784e-16	6.661338e-14
199	20681204	0.251855	2.790954	2.790954	2.358786	2.358786	4.440892e-15	-8.437695e-14
200	20690304	0.250379	2.788826	NaN	NaN	NaN	NaN	NaN

201 rows × 8 columns

In [ ]:

In [ ]:

Q8&9 Parametric models

In [11]:
```
# Nelson and Siegel & Svensson

class ParamsYieldCurve(BaseYieldCurve):
    def __init__(self, input_date, start_date, input_path, end_date):
        BaseYieldCurve.__init__(self, input_date, start_date, input_path, end_date)
        self.input_data["spot_rate"] = np.nan
        self.input_data["fitted_Mid"] = np.nan

    def term_struc_func(self, T, x):
        # spot rate r(t, T)
        pass

    def optimize_all_params(self, x):
        self.input_data["spot_rate"] = self.term_struc_func(
            self.input_data["ACT/360"], x)

        self.input_data["discount_factor"] = np.exp(
            -self.input_data["spot_rate"]*self.input_data["ACT/360"])

        cond_libor_discount = (self.input_data["Rate Type"]=="Cash Rates")
        self.input_data.loc[cond_libor_discount, "fitted_Mid"] = \
                    ((1/self.input_data.loc[cond_libor_discount, "discount_factor"])-1)/ \
                        (self.input_data.loc[cond_libor_discount, "ACT/360"])
        self.input_data.loc[cond_libor_discount, "fitted_Mid"] = 100 * \
                self.input_data.loc[cond_libor_discount, "fitted_Mid"]

        cond_fwd = (self.input_data["Rate Type"]=="Contiguous Futures")
        for fwd_end_date in self.input_data.loc[cond_fwd, "Maturity_date"]:

            fwd_end_idx = self.input_data.loc[self.input_data["Maturity_date"]==fwd_end_date].index[0]

            fwd_start_date = self.fwd_dates[self.fwd_dates<fwd_end_date].iloc[-1]
            fwd_start_idx = self.input_data.loc[self.input_data["Maturity_date"]==fwd_start_date].index[0]

            foward_discount = self.input_data.loc[fwd_end_idx, "discount_factor"] / \
                    self.input_data.loc[fwd_start_idx, "discount_factor"]

            self.input_data.loc[fwd_end_idx, "fitted_Mid"] = ((1/foward_discount)-1)/ \
                (self.input_data.loc[fwd_end_idx, "ACT/360"])
        self.input_data.loc[cond_fwd, "fitted_Mid"] = 100 * \
                self.input_data.loc[cond_fwd, "fitted_Mid"]
```

```
        swap_data_idxs = self.input_data.loc[self.input_data["Rate Type"]=="Swap Rates"].index
        for i, swap_end_idx in enumerate(swap_data_idxs):
            swap_end_date = self.input_data.loc[swap_end_idx, "Maturity_date"]

            coupon_dates = self.swap_dates[self.swap_dates<=swap_end_date][1:]
            freq = self.input_data.loc[swap_end_idx, "Freq"]

            df_coupons = copy.deepcopy(self.input_data.loc[
                                self.input_data["Maturity_date"].isin(coupon_dates)])
            swap_yield = freq*(1-df_coupons.loc[swap_end_idx, "discount_factor"])/(df_coupons["discount_facto
r"].sum())
            self.input_data.loc[swap_end_idx, "fitted_Mid"] = swap_yield
        self.input_data.loc[swap_data_idxs, "fitted_Mid"] = 100 * \
                self.input_data.loc[swap_data_idxs, "fitted_Mid"]

        df_y_yhat = self.input_data.loc[
            self.input_data["Term"].notnull(), ["Mid", "fitted_Mid"]]
        return ((df_y_yhat["Mid"] - df_y_yhat["fitted_Mid"])**2).sum()


    def fit_curve(self, init_params):
        res = minimize(self.optimize_all_params, init_params)
        return res.x

    def get_final_output(self, fitted_params):

        df_final_output = self.df_output
        df_final_output["spot_rate"] = self.term_struc_func(df_final_output["ACT/360"], fitted_params)
        df_final_output["discount_factor"] = np.exp(-df_final_output["spot_rate"]*df_final_output["ACT/360"])
        df_final_output.loc[0, "discount_factor"] = 1.

        fwd_discount = df_final_output["discount_factor"].shift(-1) / df_final_output["discount_factor"]
        df_final_output["forward_rate"] = ((1/fwd_discount)-1)/ \
                (df_final_output["ACT/360"].shift(-1) - df_final_output["ACT/360"])
        df_final_output["forward_rate"] = df_final_output["forward_rate"]*100

        # semi-annually compounded zero rate
        df_final_output["zero_rate"] = (np.power((1/df_final_output["discount_factor"]),
                                        1/(df_final_output["ACT/360"]*2))-1)*2
        df_final_output["zero_rate"] = df_final_output["zero_rate"]*100
```

```
        return df_final_output
```

In [12]:

```
class NelsonSiegelYieldCurve(ParamsYieldCurve):
    def __init__(self, input_date, start_date, input_path, end_date):
        ParamsYieldCurve.__init__(self, input_date, start_date, input_path, end_date)

    def term_struc_func(self, T, x):
        tao_1, tao_2, beta_0, beta_1, beta_2 = x
        return beta_0 + beta_1 * ((1-np.exp(-T/tao_1))/(T/tao_1)) + \
                beta_2 * ((1-np.exp(-T/tao_2))/(T/tao_2) - np.exp(-T/tao_2))


class SvenssonYieldCurve(ParamsYieldCurve):
    def __init__(self, input_date, start_date, input_path, end_date):
        ParamsYieldCurve.__init__(self, input_date, start_date, input_path, end_date)

    def term_struc_func(self, T, x):
        tao_1, tao_2, tao_3, beta_0, beta_1, beta_2, beta_3 = x
        return beta_0 + beta_1 * ((1-np.exp(-T/tao_1))/(T/tao_1)) + \
                beta_2 * ((1-np.exp(-T/tao_2))/(T/tao_2) - np.exp(-T/tao_2)) + \
                    beta_3 * ((1-np.exp(-T/tao_3))/(T/tao_3) - np.exp(-T/tao_3))
```

In [ ]:

In [13]:

```
nelson_yield_curve = NelsonSiegelYieldCurve("20190228", "20190304", input_path, "20681204")
fitted_params_nelson = nelson_yield_curve.fit_curve([1.0, 5.0, 0.0, 0.0, 0.0])
print(fitted_params_nelson)

df_final_output_nelson = nelson_yield_curve.get_final_output(fitted_params_nelson)
```

```
[ 0.09082463  6.62048077  0.04488165 -0.05936117 -0.06747841]
```

In [14]:
```
df_final_output_nelson
```

Out[14]:

	Maturity_date	ACT_days	30I_days	ACT/360	30I/360	is_keep	spot_rate	discount_factor	forward_rate	zero_rate
0	20190304	0	0	0.000000	0.000000	1	NaN	1.000000	2.385316	0.000000
1	20190604	92	90	0.255556	0.250000	1	0.023781	0.993941	4.021781	2.392269
2	20190904	184	180	0.511111	0.500000	1	0.031897	0.983829	3.910764	3.215235
3	20191204	275	270	0.763889	0.750000	1	0.034219	0.974199	3.712469	3.451370
4	20200304	366	360	1.016667	1.000000	1	0.034899	0.965142	3.523262	3.520488
...	...	...	...	...	...	...	...	...	...	...
196	20680305	17899	17641	49.719444	49.002778	1	0.035830	0.168395	4.486114	3.615275
197	20680604	17990	17730	49.972222	49.250000	1	0.035874	0.166507	4.487297	3.619796
198	20680904	18082	17820	50.227778	49.500000	1	0.035919	0.164619	4.487894	3.624325
199	20681204	18173	17910	50.480556	49.750000	1	0.035962	0.162773	4.488453	3.628765
200	20690304	18263	18000	50.730556	50.000000	1	0.036005	0.160966	NaN	3.633116

201 rows × 10 columns

In [ ]:

In [15]:
```
svensson_yield_curve = SvenssonYieldCurve("20190228", "20190304", input_path, "20681204")
fitted_params_svensson = svensson_yield_curve.fit_curve([1.0, 5.0, 10., 0.0, 0.0, 0.0, 0.0])
print(fitted_params_svensson)

df_final_output_svensson = svensson_yield_curve.get_final_output(fitted_params_svensson)
```

```
[ 0.38336776  2.31193793 23.85597457  0.17183695 -0.17728454 -0.33192902
 -0.41559066]
```

In [16]:
```
df_final_output_svensson
```

Out[16]:

	Maturity_date	ACT_days	30I_days	ACT/360	30I/360	is_keep	spot_rate	discount_factor	forward_rate	zero_rate
0	20190304	0	0	0.000000	0.000000	1	NaN	1.000000	2.324889	0.000000
1	20190604	92	90	0.255556	0.250000	1	0.023180	0.994094	5.281875	2.331494
2	20190904	184	180	0.511111	0.500000	1	0.037823	0.980854	5.797164	3.818268
3	20191204	275	270	0.763889	0.750000	1	0.044351	0.966688	5.276259	4.484642
4	20200304	366	360	1.016667	1.000000	1	0.046356	0.953965	4.395595	4.689709
...	...	...	...	...	...	...	...	...	...	...
196	20680305	17899	17641	49.719444	49.002778	1	0.032143	0.202270	6.489990	3.240312
197	20680604	17990	17730	49.972222	49.250000	1	0.032306	0.199005	6.551237	3.256880
198	20680904	18082	17820	50.227778	49.500000	1	0.032473	0.195728	6.611248	3.273770
199	20681204	18173	17910	50.480556	49.750000	1	0.032638	0.192511	6.670521	3.290613
200	20690304	18263	18000	50.730556	50.000000	1	0.032804	0.189353	NaN	3.307403

201 rows × 10 columns

In [ ]: